

# Attacking the combination generator <sup>★</sup>

Frédéric Didier<sup>1</sup> and Yann Laigle-Chapuy<sup>2</sup>

<sup>1</sup> EPFL, IC - IIF - ALGO, Bâtiment BC,  
Station 14, CH - 1015 Lausanne  
`frederic.didier@epfl.ch`

<sup>2</sup> Projet SECRET, INRIA Rocquencourt, Domaine de Voluceau,  
78153 Le Chesnay cedex  
`yann.Laigle-Chapuy@inria.fr`

**Abstract.** We present one of the most efficient attacks against the combination generator. This attack is inherent to this system as its only assumption is that the filtering function has a good autocorrelation. This is usually the case if the system is designed to be resistant to other kinds of attacks. We use only classical tools, namely vectorial correlation, weight 4 multiples and Walsh transform.

**Keywords:** Stream cipher, combination generator, Boolean functions, low weight multiples, Walsh transform.

## 1 Introduction

The combination generator is, together with the filter generator, one of the simplest and most analyzed construction of stream ciphers. It uses as an internal state many linear feedback shift registers (LFSRs). We will write  $m$  for their total size in bits. These registers are filtered using a  $n$ -variable balanced Boolean function  $f$  (from  $\mathbf{F}_2^n$  into  $\mathbf{F}_2$ ) to produce the keystream  $(z_t)_{t \geq 0}$ . The inputs of this function are taken from some bits in the LFSRs internal states. We will write  $\mathbf{x}_t$  for the  $n$ -bit vector corresponding to the inputs of  $f$  at time  $t$ . Notice that we will always write such vector of bits in bold. Our goal here is to find the key (that is the initial state of all the LFSRs) knowing the keystream sequence  $(z_t)_{t \geq 0}$  and all the components of the combination generator.

The classical way to attack such a system is to use a correlation attack [Sie85] or one of its variants called fast correlation attacks [MS88,CT00,JJ00]. The idea is to exploit the existence of a statistical dependence between the keystream and one of the constituent LFSR. For example, if we write  $\mathbf{x} = (\mathbf{u}, \mathbf{v})$  where  $\mathbf{u}$  corresponds to the bits taken from the first LFSR, such a dependence exists if for some linear function  $\ell$ ,

$$\Pr[f(\mathbf{u}, \mathbf{v}) = \ell(\mathbf{u})] \neq 1/2 .$$

---

<sup>★</sup> This work is partially funded by CELAR/DGA.

One can then perform an exhaustive search on the initial state of the first LFSR (or any other targeted LFSR) and try to detect a bias. In order to ensure that there is no such bias (or a really low one)  $f$  is usually chosen with a high non-linearity and a low autocorrelation.

In the fast correlation attacks, one sees the search of the initial state as a decoding problem of the linear code generated by the target LFSR and the linear function  $\ell$ . Then, instead of an exhaustive search, it is possible to avoid examining all possible initializations of the target LFSR by using some efficient error-correcting techniques. Usually the attack becomes faster but requires a larger amount of keystream.

This paper presents a new attack on the combination generator. At first sight, it may not seem new because we use only classical tools, namely low weight multiples, vectorial correlation and Walsh transform. But in our knowledge, they are used in an original way and we add some interesting insights. Moreover, this attack is not based on any particular weakness of the filtering function like for correlation attacks. Actually, we can even attack a system where we do not know the filtering function used! To simplify our analysis we only assume that  $f$  has a good autocorrelation. However choosing a filtering function without this property will open the door to the correlation attacks described above.

The paper is organized as follows. We begin by explaining the attack principle in the next section. Then, in Section 3, we detail the algorithm and the complexity of the different steps involved. We give in Section 4 the actual time complexity of our implementation on an example combination generator. We finally conclude in the last section.

## 2 Attack principle

We describe here an attack on the combination generator based on the theorem given at the end of this section. What we actually show is how to find the initial state of some of the LFSR composing the system. This is certainly enough to say that the system is insecure and recovering the rest of the initial state require usually less work. The best method for this task is more dependent on the actual system we are working on and we will give in Section 4 an example of how we can actually do it.

So, suppose we are given a combination generator and we are able to observe a large amount of keystream bits, the  $z_t$ 's. We denote by  $\mathbf{x}_t$  the input of the filtering function for this observed system at time  $t$ , that is we have  $z_t = f(\mathbf{x}_t)$ . We split the LFSRs involved into two groups and do an exhaustive search on the initial states of the first LFSRs. For a given value  $I$  of this partial initial state, we write  $\mathbf{y}_t$  for the hypothetical first part of the filtering function input at time  $t$ . That is, the part we can compute given  $I$ . We also need one or more common weight 4 multiples  $1 + X^{t_1} + X^{t_2} + X^{t_3}$  of the feedback polynomials of the LFSRs in the second group. We then compute the ratio  $P_I$  of times  $t$  such that

$$z_t + z_{t+t_1} + z_{t+t_2} + z_{t+t_3} = 0$$

amongst all the  $t$ 's such that

$$\mathbf{y}_t + \mathbf{y}_{t+t_1} + \mathbf{y}_{t+t_2} + \mathbf{y}_{t+t_3} = \mathbf{0} .$$

Remark that the number of such time instants at our disposal is directly linked to the amount of keystream we know. Now, how can we distinguish the real partial initial state from the others?

- If  $I$  is the actual first part of the initial state, then we know we look only at times  $t$  such that  $\mathbf{x}_t + \mathbf{x}_{t+t_1} + \mathbf{x}_{t+t_2} + \mathbf{x}_{t+t_3} = \mathbf{0}$ . This is because we are working with a multiple of the other LFSRs feedback polynomials. Then, it is reasonable to assume that the ratio  $P_I$  is an estimate of the probability  $\Pr(f(\mathbf{u}_1) + f(\mathbf{u}_2) + f(\mathbf{u}_3) + f(\mathbf{u}_4) = 0 \mid \sum_i \mathbf{u}_i = \mathbf{0})$ . Here the  $\mathbf{u}_i$  are vectors in  $\mathbf{F}_2^n$  uniformly distributed amongst the one having the required property.
- If  $I$  is not the actual first part of the initial state, even in the worst case where only the initial state of one LFSR was not guessed correctly, it is reasonable to assume that the sum  $\mathbf{x}_t + \mathbf{x}_{t+t_1} + \mathbf{x}_{t+t_2} + \mathbf{x}_{t+t_3}$  is different from  $\mathbf{0}$  roughly half of the time. For these points, we then have an estimate for one of the probabilities  $\Pr(f(\mathbf{u}_1) + f(\mathbf{u}_2) + f(\mathbf{u}_3) + f(\mathbf{u}_4) = 0 \mid \sum_i \mathbf{u}_i = \mathbf{u})$  where  $\mathbf{u} \neq \mathbf{0}$ .

Hopefully for us, the two cases are distinguishable thanks to the following result taken almost directly from [Did07] and already present in the work of Sabine Leveiller [Lev04].

**Theorem 1.** *Let  $f$  be a  $n$ -variable balanced Boolean function, and let  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4$  be 4 uniformly distributed  $n$  bits vector such that  $\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 + \mathbf{u}_4 = \mathbf{u}$ . Let  $P_{\mathbf{u}} := \Pr(f(\mathbf{u}_1) + f(\mathbf{u}_2) + f(\mathbf{u}_3) + f(\mathbf{u}_4) = 0)$ , then we have*

$$P_{\mathbf{0}} \geq \frac{1}{2} + \frac{1}{2^{n+1}}$$

and

$$\min_{\mathbf{u} \neq \mathbf{0}} (P_{\mathbf{0}} - P_{\mathbf{u}}) \geq \frac{1}{2^{n+1}} \left(1 - \frac{\Delta_f}{2^n}\right)^2$$

where  $\Delta_f$  is the maximum of the autocorrelation coefficients of  $f$  and is usually small compared to  $2^n$ .

*Proof.* see Appendix.

We will assume in the rest of this paper that the filtering function has a good autocorrelation property which result in a difference between these probabilities of  $\frac{1}{2^{n+1}}$ . This is a reasonable hypothesis because it is the case for the function usually used in this settings [GK03]. In particular, it is a needed property to resist against correlation attacks.

### 3 Detailed analysis

We describe here our attack in details and hence need some more notations. Since we split the LFSRs in two groups, we will write  $m = m_1 + m_2$  where  $m_1$  is the total size in bits of the LFSRs in the first group. That is the one we do an exhaustive search on. Similarly, we write  $n = n_1 + n_2$  for the inputs bit of the filtering function  $f$ . Remark that in practice,  $n$  is kept small compared to  $m$  for efficiency issues. Our result is summarized in this theorem:

**Theorem 2.** *Our attack recovers  $m_1$  bits of the initial states in complexity  $O(m_1 2^{n_1} 2^{m_1})$ . It requires  $O(2^{\frac{m_2}{3}} + m_1 2^{2n+n_1+1})$  consecutive bits of keystream and a memory of  $O(2^{m_1})$ . If the memory requirement is too big, some tradeoff exists until a complexity of  $O(m_1 2^{2n+n_1} 2^{m_1})$  and a memory of  $O(m_1 2^{2n+n_1})$ . This attack also require a precomputation phase of complexity and memory in  $O(2^{\frac{m_2}{3}})$ .*

#### 3.1 Computing weight 4 multiples

We need to compute one or a few weight 4 common multiples of the second group of LFSRs. We show here that this precomputation phase can be dealt with a complexity and memory around  $O(2^{\frac{m_2}{3}})$ .

What we need is only a few multiple of degree as small as possible. If we look at LFSRs of total size  $m_2$ , it is well known that the expected number of weight 4 multiples of degree  $D$  is heuristically approximated by  $\frac{1}{2^{m_2}} \frac{D^3}{6}$  considering that for  $D$  large enough the values of the polynomials of weight 4 and degree at most  $D$  are uniformly distributed.

There are many algorithms to compute low weight multiples whose complexity depends on the parameters  $D$  which in our case is around  $2^{\frac{m_2}{3}}$ . One can use the algorithm in [CJM02] but for weight 4 multiples, the most efficient is the one of [DLC07]. It's especially adapted here, as we need to find a simultaneous multiple of many polynomials.

Its complexity is in  $O(D)$  times the complexity to compute discrete logarithms in the multiplicative group  $\mathbf{F}_{2^{l_1}}^* \times \dots \times \mathbf{F}_{2^{l_k}}^*$  where the  $l_i$  are the respective length of each LFSR. This task is particularly easy since using Pohlig-Hellman algorithm it can be splitted into  $k$  discrete logarithms computation in each of the finite field multiplicative groups involved.

#### 3.2 Amount of keystream needed

We prove here the following lemma

**Lemma 1.** *We need to consider around  $N := m_1 2^{2n+n_1+1}$  degree 4 equations to identify the correct partial initial state. This translates to  $O(2^{\frac{m_2}{3}} + m_1 2^{2n+n_1+1})$  consecutive keystream bits needed.*

Given the results in Section 2, mainly Theorem 1, we need to distinguish in the worst case between two binomial distributions, one of parameter  $\frac{1}{2} + \frac{1}{2^{n+1}}$  and one of parameter at most  $\frac{1}{2} + \frac{1}{2^{n+2}}$ . However, for most of the wrong partial initial states, we will just observe a law of parameter  $\frac{1}{2}$ , so the bias we need to detect is in practice close to  $\frac{1}{2^{n+1}}$ .

A classical results from statistics tell us that using  $S$  samples, we have an error probability of roughly  $2^{-\frac{S}{2^{n+1}}}$ . We thus need at least  $2^{2n+1}$  samples to be able to distinguish the two distributions. But this is not sufficient in our case. Recall that we are doing an exhaustive search on  $2^{m_1}$  possible states, so the average number of wrong states passing the statistical test will be  $2^{m_1} 2^{\frac{S}{2^{n+1}}}$ . We thus need a number of samples equal to at least  $m_1 2^{2n+1}$  to obtain only a few possible candidates for the real initial states. Notice that with this number of samples, the probability to miss the initial states is of  $O(2^{-m_1})$  which is really small.

What is the amount of keystream needed to get that many samples? For one sample, we will need to consider  $2^{n_1}$  degree 4 equations since this is the expected number for  $\mathbf{y}_t + \mathbf{y}_{t+t_1} + \mathbf{y}_{t+t_2} + \mathbf{y}_{t+t_3}$  to be equal to  $\mathbf{0}$ . The average degree of the lowest weight 4 multiples is  $O(2^{\frac{m_2}{3}})$  (see previous subsection). By shifting this multiple  $x$  times, we get  $x$  degree 4 equation for  $O(2^{\frac{m_2}{3}} + x)$  consecutive keystream bits. Hence, the required amount of keystream is  $O(2^{\frac{m_2}{3}} + m_1 2^{2n+n_1+1})$ .

We assumed here that we use only one weight 4 multiples. If we use many, it is actually possible to need less keystream at the expense of more precomputation. This gain is difficult to analyze as it really depend on the used LFSRs, but may definitely be useful in practice. However, it will not change the overall asymptotic keystream needed.

### 3.3 Performing the attack efficiently

We show in this subsection how to find the initial states in  $O(m_1 2^{n_1} 2^{m_1})$  instead of  $O(N 2^{m_1})$  with a straightforward implementation. Notice that this is a huge gain since in our case  $m_1 2^{n_1} \ll N$ . On the memory side, we need  $2^{m_1}$  integer in the second case compared to  $N$  bits in the first one. If this is a problem, we can actually trade memory for time and be anywhere between those two algorithms.

In order to perform our attack, we have to compute many quantities involving time positions of the form  $(t, t+t_2, t+t_2, t+t_3)$ . Since we may use many multiples, let just assign an index  $i$  to such 4-tuple. We will then write  $z(i)$  for the sum of the  $z_t$  for the 4 time positions number  $i$  and  $\mathbf{y}(i)$  in the same way.

Computing the probability estimate for a given partial initial state is roughly the same as computing the number of indices  $i$  such that  $\mathbf{y}(i) = \mathbf{0}$  and  $z(i) = 0$ . Actually to get the true probability, we also need to know how many indices are such that  $\mathbf{y}(i) = \mathbf{0}$  and  $z(i) = 1$  but this will not change our discussion or the final complexity. So let restrict ourselves on the  $N'$  indices  $i$  such that  $z(i) = 0$ .

If we do this independently for each of the  $2^{m_1}$  partial initial states, the complexity is then in  $N' 2^{m_1}$  which is pretty large. In order to improve on this

complexity, let us start by assuming that out of the  $m_1$  bits, only one is used as an input for the filtering function  $f$ , that is we assume the  $\mathbf{y}_t$  to be scalar. Remark now that any linear combination of bits from the internal states of some LFSRs can be expressed as a linear expression of the initial state of these LFSRs. This is the case for the  $\mathbf{y}(i)$ . We can then define a binary linear code of generator matrix  $G$  of size  $m_1 \times N'$  such that for a given partial initial state  $\mathbf{u}$  the  $i$ -th element of  $\mathbf{u}G$  is precisely  $\mathbf{y}(i)$ . The number we try to compute for a given initial state  $\mathbf{u}$  is then just  $N'$  minus the Hamming weight of  $\mathbf{u}G$ .

Is there a way to compute the Hamming weight of each codeword in a code of length  $N'$  and dimension  $m_1$  faster than  $2^{m_1}N'$ ? The answer is yes, thanks to the Walsh transform we can do it in  $O(m_1 2^{m_1})$  which in our case is a lot better since  $m_1 \ll N'$ . The Walsh transform  $\hat{w}$  of a function  $w : \mathbf{F}_2^{m_1} \rightarrow \mathbf{Z}$  can be computed in  $O(m_1 2^{m_1})$  and is such that

$$\hat{w}(\mathbf{u}) = \sum_{\mathbf{v}} w(\mathbf{v}) (-1)^{\mathbf{v} \cdot \mathbf{u}} .$$

If  $w(\mathbf{v})$  is equal to the number of columns in  $G$  equal to  $\mathbf{v}$ ,  $\hat{w}(\mathbf{u})$  is exactly twice the Hamming weight of  $\mathbf{u}G$  minus  $N$ .

Now, what to do when the  $\mathbf{y}_t$  are not scalar? we can use this nice formula :

$$\#\{i, \mathbf{y}(i) = \mathbf{0}\} = \frac{\sum_{\mathbf{y} \in \mathbf{F}_2^{n_1}} \#\{i, \mathbf{y}(i) \cdot \mathbf{y} = 0\} - 2^{n_1-1}}{2^{n_1}} .$$

This simply comes from the fact that when  $\mathbf{y}(i)$  is  $\mathbf{0}$  it contributes to  $2^{n_1}$  in the sum whereas any other  $\mathbf{y}(i)$  contributes only  $2^{n_1-1}$ . Each cardinality in the sum can be computed using a Walsh transform, but since Walsh transform is linear, we better compute directly the Walsh transform of the function

$$w(\mathbf{v}) = \#\{(i, \mathbf{y}), \mathbf{y} \cdot (G_1(i), \dots, G_{n_1}(i)) = \mathbf{v}\}$$

where  $G_j(i)$  is the  $i$ -th columns of the matrix of the linear code corresponding to the input bit  $j$  of the filtering function.

Finally, the time-memory tradeoff mentioned in the first paragraph directly follows from a time memory tradeoff in the implementation of the Walsh transform.

## 4 Attack Example

We give in this section an example of how to use our result to mount an attack on a given combination generator.

The following timings were obtained on an Intel Core2 Quad CPU Q9550 at 2.83GHz, using only one core and no more than 2GB of memory. We used a combination generator based on three LFSRs of size 29, 31 and 37 respectively. The feedback polynomials are dense in order not to have artificially easy to find low weight multiples. The filtering function  $f$  is a 9-variable Boolean function, with 3 inputs from each LFSR. It was chosen to be still balanced even if we fixed

	Precomputation	Total online time	Keystream used
Attack 1	12min27s	7min01s	3.06MB
Attack 2	3min02s	6h18min	985KB

**Table 1.** Global comparison of the two attacks

	1st LFSR	2nd LFSR	3rd LFSR
Attack 1	51s	6min10s	0s
Attack 2	6h17min	1min27s	0s

**Table 2.** Comparison of the different parts of the online attacks

the input bits from any of the constituent LFSR in order to avoid traditional correlation attack , i.e. the function is 3-resilient.

In the attack referred below as Attack 1, we retrieve the initialization of the LFSR of size 29, 31 and 37 in this order. We thus have in a first step the following parameters :

$$m_1 = 29; m_2 = 31 + 37 = 68; n_1 = 3; n_2 = 6.$$

As stated in Lemma 1, we thus need to consider approximately  $N = 2^{26.86}$  multiples of  $P_{31} \times P_{37}$  of weight 4. The maximum degree needed is slightly less than  $2^{25}$  which implies we need 3MB of keystream. Using the approach of [DLC07] for finding low weight multiples of degree less than  $2^{25}$ , the precomputation took around 13 minutes. The online time to recover the initial internal state of the first LFSR is only 51s, for a theoretical workload of approximately  $2^{37}$ . The internal state of the second LFSR is then recovered in roughly 6 minutes, using only very few keystream because we only need to consider multiples of the last feedback polynomial. Finally, the internal state of the last register is found by a different method almost instantly, because the full knowledge of 6 entries of our Boolean function amongst 9 gives us a lot of information.

In the second version of the attack , we retrieve the initialization of the LFSR of size 37, 29 and 31 in this order. We thus have in a first step the following parameters :

$$m_1 = 37; m_2 = 31 + 29 = 60; n_1 = 3; n_2 = 6.$$

This small difference allows us to have a lower value for  $m_2$ , which implies that we need less keystream to perform the attack. We need to consider approximately  $N = 2^{26.86}$  multiples of  $P_{31} \times P_{29}$  of weight 4, the precomputation is this time only 3 minutes and the maximum degree needed is approximately  $2^{23}$ , corresponding to 985KB. On the contrary, the online time to recover the initial internal state of the first LFSR is much longer. The theoretical workload is approximately  $2^{42}$  and our experiment confirms this ratio as it tooks us just more than 6 hours. The timings to recover the two other LFSR is negligible compared to the first one.

In conclusion, the Attack 1 is the best one from a complexity point of view, whereas the Attack 2 minimizes the amount of keystream needed. We summarize the timings for the two different strategies in Tables 1 and 2.

## 5 Conclusion

We presented in this paper an efficient attack on the combination generator. In particular, if we look at the timings given in Section 4, we are not aware of any other attacks that can break the chosen combination generator that efficiently. Remark however that breaking the combination generator still requires an exponential number of computation. Hence, if the parameters are chosen large enough, such a system can still be secure given the actual knowledge.

An important point is that the presented attack is inherent to the construction and is not based on any particular weakness in the choice of the filtering function or in the constituent LFSRs. As such, it appears that for the same order of internal size the combination generator is a lot less secure than a single large primitive LFSR filtered by a non-linear function. This statement seems true in many ways. With a large LFSR, it is more difficult to compute any low weight multiples, it is more difficult to break the system into smaller components, and the best attacks we know are a lot less efficient.

## Acknowledgment

We would like to thank Anne Canteaut and Jean-Pierre Tillich for their help and their useful comments.

## References

- Can06. Anne Canteaut. *Analyse et conception de chiffrements à clef secrète*. Habilitation à diriger des recherches, Université de Paris 6, 2006.
- CJM02. P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: an algorithmic point of view. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.
- CT00. Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *EUROCRYPT*, pages 573–588, 2000.
- Did07. Frédéric Didier. Attacking the filter generator by finding zero inputs of the filtering function. *Indocrypt 2007*, 2007.
- DLC07. Frédéric Didier and Yann Laigle-Chapuy. Finding low-weight polynomial multiples using discrete logarithm. *IEEE International Symposium on Information Theory - ISIT 2007*, 2007.
- GK03. Guang Gong and Khoongming Khoo. Additive autocorrelation of resilient boolean functions. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 275–290. Springer, 2003.



- JJ00. Thomas Johansson and Fredrik Jöhansson. Fast correlation attacks through reconstruction of linear polynomials. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 300–315, London, UK, 2000. Springer-Verlag.
- Lev04. Sabine Leveiller. *Quelques algorithmes de cryptanalyse du registre filtré*. PhD thesis, Télécom Paris, ENST, November 2004.
- MS88. W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Advances in Cryptology - EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer-Verlag, 1988.
- Sie85. Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Computers*, 34(1):81–85, 1985.

## A Proof of Theorem 1

The bound on  $P_0$ , mentioned in [Can06], is already present in [Lev04]. It is a direct consequence of the result in [Did07] where it is shown that

$$P_{\mathbf{x}} = \frac{1}{2} \left( 1 + \sum_{\mathbf{y} \in \mathbf{F}_2^n} (-1)^{\mathbf{y} \cdot \mathbf{x}} \left( \frac{W_f(\mathbf{y})}{2^n} \right)^4 \right)$$

where  $W_f(\mathbf{y})$  is the Walsh coefficient of  $f$  at point  $\mathbf{y}$ , that is  $\sum_{\mathbf{x}} (-1)^{f(\mathbf{x}) + \mathbf{x} \cdot \mathbf{y}}$ . By Parseval equality we know that the sum  $\sum_{\mathbf{y}} W_f(\mathbf{y})^2$  is equal to  $2^{2n}$  and it is well known that the sum of square  $\sum_{\mathbf{y}} (W_f(\mathbf{y})^2)^2$  is minimized when every term is equal. Hence we can upper bound everything by

$$\frac{1}{2} \left( 1 + \frac{1}{2^{4n}} \sum_{\mathbf{y} \in \mathbf{F}_2^n} (2^n)^2 \right)$$

and get the first part of the theorem.

For the second part, the result is taken directly from [Did07]. Just recall that the maximum of the autocorrelation coefficient is defined as

$$\Delta_f := \max_{\mathbf{y} \neq \mathbf{0}} \left| \sum_{\mathbf{x} \in \mathbf{F}_2^n} (-1)^{f(\mathbf{x}) + f(\mathbf{x} + \mathbf{y})} \right|$$

and is usually quite small for Boolean functions used in cryptography.